Einführung in den Forschungscode

1 objektorientierte Programmierung

Bei der objektorientierten Programmierung (OOP) wird alles in Objekten gespeichert. Beispielsweise auch Variablen, diese müssen dann nicht mehr einzeln an andere Dateien/Funktionen übergeben werden, sondern es reicht aus das passende Objekt zu übergeben. Die Objekte gehören einer bestimmten Klasse an. In dieser Klasse sind alle Attribute (Eigenschaften) und Methoden (Funktionen) definiert, die ein Objekt haben/ausführen kann. Ein Objekt ist die Instanz einer Klasse und kann mit anderen Objekten kommunizieren durch senden und empfangen von Nachrichten.

1.1 Klasse

Eine Klasse ist ein sogenannter Bauplan für eine bestimmte Gruppe von Objekte. Sie enthält Methoden (Funktionen) und Attribute (Eigenschaften/Konfigurationsparameter). Die Gruppierung von Objekten ist möglich durch die Zugehörigkeit zu einer bestimmten Klasse.

Wenn mehrere Objekte sich nur in wenigen Eigenschaften unterscheiden, aber der restliche Aufbau gleich ist, kann mit Vererbung gearbeitet werden. Dass bedeutet, die gleichen Eigenschaften werden in einer Basisklasse (engl. superclass) definiert. Die sich unterscheidenden Eigenschaften werden dann in Unterklassen definiert, die von der Basisklasse abgeleitet sind. So erbt die Unterklasse die Datenstruktur der Basisklasse und ergänzt sie entsprechend.
Syntax:

```
classdef Unterklasse < Basisklasse
    properties
        ...
    end
    methods
        ...
    end
end
```

1.2 Objekt

Wie bereits erwähnt kann auf Grundlage der Klassendefinition ein Objekt erstellt werden. Dabei beschreiben die Attribute/Eigenschaften den Zustand des Objekts. Die Methoden/Funktionen sind hingegen sämtliche Aktionen, die ein Benutzer ausführen kann. Objekte sind immer in einem wohldefinierten, selbstkontrollierten Zustand. Um zu überprüfen aus welcher Klasse ein Objekt stammt:

```
>> class(Objekt)
```

1.3 Attribute

Die Attribute sind die Eigenschaften eines Objekts. Wer Zugriff auf diese Eigenschaften hat, kann in der zugehörigen Klasse definiert werden:

- Get Access = private: Eigenschaft nur sichtbar für Methoden, die mit ihr arbeiten (schreibgeschützt)
- constant: keine Änderung der Eigenschaft möglich
- dependent: Eigenschaft wird nur berechnet, wenn sie angefordert wird → Get-Methode angeben, die beim Zugriff auf die Eigenschaft automatisch aufgerufen wird

Syntax:

properties
    ...
end
properties(SetAccess = private)
    ...
end
properties(Constant)
    ...
end
properties(Dependent = $true$)
    ...
end

1.4 Methoden

Methoden sind Funktionen, die in einer bestimmten Klasse definiert sind. Auf diese Funktionen können je nach Zugriffsebene unterschiedlich viele Objekte zugreifen und diese ausführen. Im Forschungscode wurden drei Zugriffsebenen verwendet:

- öffentliche (public) Methoden: dürfen von allen Klassen und deren Instanzen aufgerufen werden
- geschützte (protected) Methoden: dürfen von Klassen im selben Paket und abgeleiteten Klassen aufgerufen werden
- private (private) Methoden: können nur von anderen Methoden derselben Klasse aufgerufen werden

Syntax:

methods
  function $ordinaryMethod(obj, arg1, ...)$
      ...
  end
end

Konstruktormethoden dienen der Dateninitialisierung und -validierung eines Objekts.
Syntax:

```
methods
  function obj = set.PropertyName(obj, value)
      ...
  end
  function value = get.PropertyName(obj)
      ...
  end
end
```

1.5 Ziel/Vorteile

- Vererbung: Dopplungen im Forschungscode vermeiden
- Polymorphie: gleiche Nachricht löst bei unterschiedlichen Objekten entsprechend andere Aktionen aus
- Datenkapselung: keine unkontrollierte Änderung des Objekts von außen
- bessere Strukturierung des Codes

2 Aufbau des Forschungscodes

Der Forschungscode ist in mehrere Ordner unterteilt. In jedem dieser Ordner sind Klassen definiert und/oder Funktionen. Die wichtigsten Matlab-Dateien zum Verständnis des Codes sind in den einzelnen Unterpunkten aufgeführt und ihre Funktionen dort erläutert. Die Überschriften der einzelnen Abschnitte entsprechen den Ordnern im Forschungscode.

2.1 commonFunctions

- *callElements.m* öffnet die richtige Elementroutine mithilfe des dofObjects, des setupObject und der feval-Funktion (siehe Erklärung weiter unten).
- *lagrangeShapeFunctions.m* beinhaltet die Formfunktionen für 1D, 2D und 3D und Gaußpunkte mit Gewichten

2.2 continuumClasses

- @neumannClass
  - *neumannClass.m* notwendige Eigenschaften und Methoden zur Berechnung eines Neumannrandes sind hier implementiert.
  - *deadLoad.m* beinhaltet die Berechnung eines Neumannrandes in 3D.
- @dirichletClass
  - *dirichletClass.m* Berechnung des Dirichletrandes ist hiermit möglich. Alle relevanten Informationen in Form einer Klasse gespeichert. Das zugehörige Objekt ist auch jeweils im dofObject enthalten (Eigenschaft: listContinuumObjects)
- @solidClass
  - *solidClass.m*

  - *displacementHookeEndpoint.m* enthält die Elementroutinen zur Berechnung mit linearelastischem Ansatz (hier mit dem impliziten Eulerverfahren). Diese Elementroutinen bestehen im Wesentlichen aus der Gaußschleife.

- @solidViscoClass
  - *solidViscoClass.m* enthält die zusätzlich notwendigen Eigenschaften und Methoden für Viskoelastizität.
  - *displacementHookeMidpoint.m* hier sind die passenden Elementroutinen auf Gaußpunktebene abgespeichert.
- weitere Klassen wie die @solidViscoClass und @solidClass sind in dem Ordner abgespeichert und sind alle von der solidSuperClass abgeleitet.

2.3 coreClasses

core = Kern, Ader

- @solidSuperClass
  - *solidSuperClass* definiert notwendige Attribute und Methoden, die für alle continuumClasses gleich sind und daher an diese vererbt werden.
- *dofClass.m* in dieser Klasse werden Attribute wie Verschiebungen und Geschwindigkeiten der einzelnen Knoten definiert und initialisiert. Die Objekte dieser Klasse beinhalten also zu jedem Zeitschritt die Verschiebungen und Geschwindigkeiten der Knoten zum Zeitpunkt t (uN und vN) und zum Zeitpunkt $t+1$ ($uN1$ und $vN1$).
- *plotClass.m* definiert die Grafikeinstellungen.
- *shapeFunctionClass.m* beinhaltet alle Ansatzfunktionen und deren Ableitungen (für 1D, 2D und 3D).
- *storageFEclass.m* ermöglicht unteranderem das Abspeichern der elementweise ausgerechneten Tangenten (Ke) und Residuen (Re).

2.4 meshes

mesh = Netz, Gitter
Hier sind alle Funktionen untergebracht, die das gegebene Gebiet diskretisieren.

- meshGenerator: z.B. *netzdehnstab.m*

2.5 scripts

- pre: z.B. *cooksMembrane.m*, *oneDimensionalContinuum.m*
- solver: z.B. *runNewton.m*, *solveScript.m*

Anhand von der Datei „*oneDimensionalContinuum.m*“ ist die Abfolge der einzelnen Funktionen und Klassen bei Ausführung des Programms genauer beschrieben.

3 wichtige Funktionen/Befehle in MATLAB

3.1 assert

„throw error if condition false“
Mithilfe dieser Funktion kann eine bestimmte Bedingung überprüft werden.

$$assert(cond)$$

Wenn die Bedingung *cond* (z.B. $x < 1$) nicht eingehalten ist, wird eine Fehlermeldung ausgerufen.

3.2 strcmp

„compare strings“
Hier werden zwei Zeichenketten verglichen. Wenn sie identisch sind, ist der 'output' 1 (true), ansonsten 0 (false).

$$tf = strcmp(s1, s2)$$
$$tf = 1 \text{ (true)} \text{ oder } 0 \text{ (false)}$$

$s1$ und $s2$ können eine Kombination aus Zeichenketten und Vektoren sein. Es werden die einzelnen Einträge an gleicher Stelle miteinander verglichen, wenn die Zeichenketten beide mehr als einen Eintrag haben.

3.3 strcmpi

„compare strings (case insensitive)“
Wie vorherigen Abschnitt werden zwei Zeichenketten miteinander verglichen. Hier wird zwischen Groß- und Kleinschreibung nicht unterschieden. Zwischen Leerzeichen hingegen schon.

$$tf = strcmpi(s1, s2)$$
$$tf = 1 \text{ (true)} \text{ oder } 0 \text{ (false)}$$

3.4 ischar

„determine if input is character array“
Es wird überprüft, ob das „input argument“ ein „character array“ ist.

$$tf = ischar(A)$$
$$tf = 1 \text{ (true)} \text{ oder } 0 \text{ (false)}$$

3.5 isfield

„determine if input is structure array field“
Diese Funktion überprüft, ob ein $field$ in einem „structure array“ enthalten ist.

$$TF = isfield(S, field)$$
$$TF = 1 \text{ (true)} \text{ oder } 0 \text{ (false)}$$

Wenn $field$ der Name eines Feldes aus dem „structure array“ S ist, ist $TF = 1$. Sonst, gibt die Funktion 0 zurück.

3.6 isnumeric

„determine whether input is numeric array“
Es wird überprüft, ob das „input argument“ ein „array of numeric data type“ ist.

$$tf = isnumeric(A)$$
$$tf = 1 \text{ (true) oder } 0 \text{ (false)}$$

Numeric types in MATLAB: int8, int16, int32, int64, uint8, uint16, uint32, uint64, single, and double.

3.7 isprop

„True if property exists“
Wenn die Eigenschaft (unter $propertyName$ abgespeichert) existiert in dem Objekt, gibt die Funktion eine 1 wieder, sonst 0.

$$tf = isprop(obj, PropertyName)$$
$$tf = 1 \text{ (true) oder } 0 \text{ (false)}$$

3.8 vertcat

„Concatenate arrays vertically“
Mithilfe dieser Funktion können mehrere Matrizen in einer zusammengefasst untereinander geschrieben werden, sofern deren Anzahl an Spalten übereinstimmt.

$$C = vertcat(A, B)$$
$$C = vertcat(A1, A2, \ldots, An)$$

Im ersten Fall wird B unter A geschrieben, diese Funktion ist äquivalent zu $[A; B]$. Es können auch mehrere Matrizen vereint werden, sofern deren erste Dimension übereinstimmt, wie in Zeile 2 gezeigt.

3.9 full

„Convert sparse matrix to full storage“
Durch diese Funktion kann eine „sparse matrix“ in eine Matrix mit allen Einträgen überführt werden.

$$A = full(S)$$

Hier ist S die „sparse matrix“ und A die volle Matrix.

3.10 strcat

„concatenate strings horizontally“
Zeichenfolgen horizontal verketten

$$s = strcat(s1, ..., sN) \rightarrow s =' s1, ..., sN'$$

3.11 isa

„determine if input has specified data type“
Entspricht A einem bestimmten Datentyp?

$$tf = isa(A, \text{dataType})$$
$$tf = 1 \text{ (true) oder } 0 \text{ (false)}$$

Wenn A ein Objekt ist, dann wird $tf = 1$, wenn „dataType“ entweder die Klasse oder Basisklasse von A.

3.12 isafield

„returns 1 if field is the name of a field of the structure array S“
Hiermit wird herausgefunden, ob im array S ein bestimmter Datentyp mit gleichem Namen vorhanden ist.

$$tf = isafield(S, (\text{field})$$
$$tf = 1 \text{ (true) oder } 0 \text{ (false)}$$

3.13 feval

„evaluate function“
Mithilfe dieser Funktion kann das erste „input argument“ als Funktion mit den darauffolgenden „input arguments“ aufgerufen werden, um „output arguments“ zu generieren

$$[y_1, ..., y_N] = feval(fun, X_1, ..., X_M)$$

Es wird die Funktion fun mit den Parametern $X_1, .., X_M$ aufgerufen und es werden dann die Parameter $Y_1, ..., Y_N$ berechnet.

3.14 numel

„number of array elements“ gibt die Anzahl der Elemente eines „arrays“ wieder

$$n = numel(A)$$

3.15 handle

Eine „handle“ ist eine Variable, die sich auf ein bestimmtes Objekt aus der handle-Klasse bezieht. Mehrere dieser handle-Variablen können sich auf ein Objekt beziehen. Sie ermöglichen es Datenstrukturen wie etwa verknüpfte Listen zu erstellen oder mit einem großen Datensatz zu arbeiten ohne ihn zu kopieren.
Die handle-Klasse ist eine abstrakte Basisklasse, mit der keine Objekte gebildet werden können. Jedoch können in Unterklassen, die von der handle-Klasse abgeleitet sind, handle-Objekte erstellt werden. Dies wurde in dem Forschungscode gemacht.

4 GitLab

4.1 Ersteinrichtung

siehe Einführung GitLab

4.2 eigenen Branch erstellen

- zum Eigenständigen und unabhängigen Arbeiten
- neuen Branch erstellen (git branch feature branch)
- checkout aus eigenem Branch (git checkout feature branch)
- add, commit und push nach dem am feature branch gearbeitet wurde

4.3 merge Request an master branch

- add, commit and push the feature branch
- switch to master branch (git checkout master)

- pull master branch to latest state (git pull)
- switch back to feature branch (git checkout feature branch)
- merge feature branch with master branch (git merge master)
- push feature branch to remote (git push)
- submit merge request via the webpage (recommend) and assign suitable reviewers

Wichtig: keine eigenen Kommentare sondern nur für alle relevante Informationen im Forschungscode / master branch

5 weiterführende Literatur

5.1 Objektorientierte Programmierung:

- https://de.wikipedia.org/wiki/Objektorientierte_Programmierung
- https://de.mathworks.com/company/newsletters/articles/introduction-to-object-oriente

5.2 GitLab

- https://git.scc.kit.edu/ifm/anleitungen/software/-/blob/master/git/simple_cheat_sheet